Assignment 7

Ariful Islam
2010576130

July 2025

1 Problem Statement

1.1 Task 1

Capture a video without any human face using a smartphone and detect objects using Ultralytics provided YOLOv8, YOLOv11 and YOLOv12 models. Compare their object detection capability, speed.

1.2 Task 2

Fine-tune any Ultralytics YOLOv8 model to prepare it as a face detector using WIDER FACE training set by splitting it into a training set and a validation set.

1.3 Task 3

Compare fine-tuned YOLOv8 based face detector with face detector available in the following GitHub directory using the validation set of WIDER FACE dataset.
https://github.com/Yusepp/YOLOv8-Face

1.4 Task 4

Build a face detector having the architecture of YOLOv1, train it using WIDERFACE WIDER FACE training set by splitting it into training set and validation set.

2 Solution

2.1 Task 1

To complete this work, we need a video, as my smartphone's camera is not so good, I took a video from online of streets of a car riding and the video is 24 seconds, and each second has multiple frames. so first we upload the video to Google Drive, then in Colab link with the drive and train the models of YOLOv8m, YOLOv11m, and YOLOV12m, and run and predict the video, and download the video. After that, we count how many instances there are and how many objects each model detects.

2.1.1 Models

All three models were downloaded from the official Ultralytics release links and run in Colab:

YOLOv8m: Medium version of YOLOv8.

YOLOv11m: Medium version of the YOLOv11 model.

YOLOv12m: Medium version of the YOLOv12 model.

Each model was used to detect objects in the entire video. After running the predictions, the processed videos were downloaded for result analysis.

2.1.2 Results and analysis

Model	Total Objects Detected
YOLOv8m	1,634
YOLOv11m	2,233
YOLOv12m	2,293

Table 1: Total number of objects detected by each YOLO model on the test video.

The results show a clear improvement in object detection performance as we move from YOLOv8m to YOLOv11m and YOLOv12m. YOLOv8m detected 1,634 objects, while YOLOv11m and YOLOv12m detected 2,233 and 2,293 objects respectively. This indicates that the newer versions (v11 and v12) are more accurate and capable of detecting more instances in the same video. YOLOv12m performed the best overall, identifying the highest number of objects. However, since YOLOv11m and YOLOv12m are larger and more complex models, they may require slightly more time and resources during inference. Overall, the comparison confirms that the updated YOLO models provide better detection accuracy, making them more suitable for real-world object detection tasks.

2.1.3 Links

- Google Colab Notebook
- GitHub Repository

2.1.4 Sample image

(a) original image

(b) v8 prediction

(c) v11 prediction

(d) v12 prediction

Figure 1: Comparison of outputs from different models

2.2 Task 2

The YOLOv8m model was fine-tuned on the WIDER FACE dataset by splitting it into training and We downloaded the WIDER FACE dataset from Kaggle and formatted it using a Python script to convert the annotations into the YOLO format. After preparing the dataset, we created a YAML configuration file specifying the dataset paths and class names.

The model chosen for fine-tuning was YOLOv8n (nano version), trained with the following parameters: image size of 640, batch size of 8, for 30 epochs, using GPU acceleration (device='cuda:0'), with a warm-up period of 5 epochs. Checkpoints were saved every 10 epochs to monitor training progress.

After training, the model achieved the following performance on the validation set: Precision: 0.833 Recall: 0.575 mAP@0.5: 0.652 mAP@0.5:0.95: 0.355

2.2.1 Results and analysis

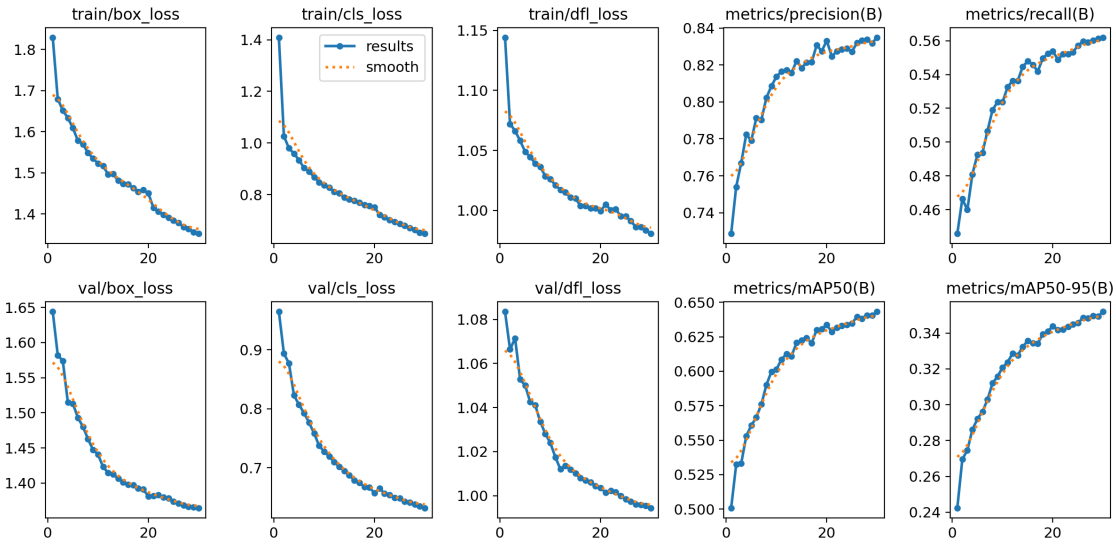

Figure 2: Result.png for the train and validation.

2.2.2 Links

- Google Colab Notebook
- GitHub Repository

2.3 Task 3

2.3.1 Comparison table

In the given GitHub repo, we can see the performance table, and if we compare with our model, we can see that our model performs less than the given repo model.

From the comparison table, we can see that the YOLOv8-Face model from GitHub performs better in terms of detection accuracy, achieving a higher mAP@0.5 score of 0.8833 on the WIDER FACE Easy validation set. In contrast, our fine-tuned YOLOv8n model achieved a lower mAP@0.5 of 0.652, which indicates that it missed more faces or had less precise bounding boxes.

Model	mAP@0.5	FPS (Avg.)	GPU Used
YOLOv8n (Fine-Tuned, Ours)	0.652	294 FPS	NVIDIA GeForce GTX 1070
YOLOv8-Face (GitHub)	0.8833 (Easy)	169 / 82 / 55 FPS	RTX 4090 / Tesla T4 / GTX 1650 Max-Q

Table 2: Comparison of fine-tuned YOLOv8n model and YOLOv8-Face on WIDER FACE validation set.

However, our model significantly outperformed in terms of inference speed, reaching 294 FPS on a GTX 1070 GPU, compared to the YOLOv8-Face model, which achieved 169 FPS (RTX 4090), 82 FPS (Tesla T4), and 55 FPS (GTX 1650 Max-Q).

The reason our model gets low accuracy is that we trained the model in less only 30 epochs, and we get higher accuracy cause our model is lighter cause to less training.

Figure 3: Ground truth value of widerface validation set image

Figure 4: Prediction batch of validation set image

2.4 Task 4

For task 2, we build yolov1 model from scratch by PyTorch implementation and train the model using the WiderFace datasets for 10 epochs. To build this model, we take help from some GitHub repo and yolov1 official paper.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 112, 112]           9,472
              ReLU-2         [-1, 64, 112, 112]               0
         MaxPool2d-3           [-1, 64, 56, 56]               0
            Conv2d-4          [-1, 192, 56, 56]         110,784
              ReLU-5          [-1, 192, 56, 56]               0
         MaxPool2d-6          [-1, 192, 28, 28]               0
            Conv2d-7          [-1, 128, 28, 28]          24,704
              ReLU-8          [-1, 128, 28, 28]               0
            Conv2d-9          [-1, 256, 28, 28]         295,168
             ReLU-10          [-1, 256, 28, 28]               0
        MaxPool2d-11          [-1, 256, 14, 14]               0
           Conv2d-12          [-1, 256, 14, 14]         590,080
             ReLU-13          [-1, 256, 14, 14]               0
           Conv2d-14          [-1, 512, 14, 14]       1,180,160
             ReLU-15          [-1, 512, 14, 14]               0
        MaxPool2d-16            [-1, 512, 7, 7]               0
           Conv2d-17            [-1, 512, 7, 7]       2,359,808
             ReLU-18            [-1, 512, 7, 7]               0
           Conv2d-19           [-1, 1024, 7, 7]       4,719,616
             ReLU-20           [-1, 1024, 7, 7]               0
           Conv2d-21           [-1, 1024, 7, 7]       9,438,208
             ReLU-22           [-1, 1024, 7, 7]               0
           Conv2d-23           [-1, 1024, 7, 7]       9,438,208
             ReLU-24           [-1, 1024, 7, 7]               0
          Flatten-25                [-1, 50176]               0
           Linear-26                 [-1, 4096]     205,524,992
             ReLU-27                 [-1, 4096]               0
          Dropout-28                 [-1, 4096]               0
           Linear-29                  [-1, 539]       2,208,283
================================================================
Total params: 235,899,483
Trainable params: 235,899,483
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 34.74
Params size (MB): 899.89
Estimated Total Size (MB): 935.20
----------------------------------------------------------------
```

Figure 5: Caption

2.4.1 Links

- Google Colab Notebook
- GitHub Repository

2.4.2 References for task 4

1. Redmon et al., You Only Look Once (YOLO): Unified, Real-Time Object Detection. arXiv:1506.02640
2. YOLOv1 PyTorch Implementation on GitHub by motokimura
3. Blog: Understanding YOLOv1 – maskaravivek.com
4. PyTorch Tutorial: Training a Classifier